Remarks

Allowance of all claims is respectfully requested. Claims 1-20 remain pending.

Initally, responsive to the 35 U.S.C. §112, second paragraph, rejection to independent claims 1, 10 & 19, Applicants herewith amend these claims to specifically accomplish what is claimed in the preamble, that is, the "transferring the object from a sender computer process to a receiver computer process." Based upon the amendements presented, reconsideration and withdrawal of the 35 U.S.C. § 112, second paragraph rejection is respectfully requested.

In the Office Action, claims 1-10 & 19 were rejected under 35 U.S.C. § 102(b) as being anticipated by Murray (U.S. Patent No. 5,944,781; hereinafter Murray), claims 1, 10 &19 were rejected under 35 U.S.C. § 102(e) as being anticipicated by Jones et al. (U.S. Patent No. 6,629,154 B1; hereinafter Jones), claims 1-4, 9-13, 18 & 19 were rejected under 35 U.S.C. § 102(e) as being anticipated by Wollrath et al. (U.S. Patent No. 6,487,607 B1; hereinafter Wollrath), and claims 5-8, 14-17 & 20 were rejected under 35 U.S.C.§ 103(a) as being unpatentable over Wollrath in view of Skinner et al. (U.S. Patent No. 6,721,740 B1; hereinafter Skinner). These rejections are respectfully, traversed to any extent deemed applicable to the claims presented herewith, and reconsideration thereof is requested for the reasons set forth below.

Applicants' recited invention is a protocol for transferring executable program code between computer processes, along with the data to be input to the executable program code. For example, claims 1 recites:

- providing an object which provides a hashtable, the hashtable having at least one set of elements, one element of the at least one set of elements comprising *executable program code,*
- wherein the executable program code includes logic *which employs as data input at least one other element of the hashtable,*
- transferring the object from a sender computer process to a receiver computer process, retrieving the executable program code from the hashtable and invoking the executable program code with the at least one other element of the hashtable as data input thereto.

Advantageously, within Applicants' recited hashtable both the executable program code and the data input to the logic of the executable program code is provided as separate elements. No similar protocol is believed taught or suggested by Murray, Jones, Wollrath or Skinner, either alone or in combination.

Murray teaches a method and apparatus for making applets persistent. A persistent applet operating on a client system may be saved along with its complete state to a remote server. When desired, the client system may retrieve the persistent applet with its previous state. (See abstract of Murray.)

In rejecting claims 1-10 & 19, the Office Action cites column 7, lines 3-9 and column 7, lines 53-62 of Murray. These lines are repeated below for convenience:

> "Objects of class "Class" are serialized as the name of the class and the fingerprint or hash of the interfaces, methods, and fields of the class. The name allows the class to be identified during deserialization and the hash of the class allows it to be verified against the class of the serialized object. All other normal Java classes are serialized by writing the encoding of its Class followed by its fields."
>
> .
>
> .
>
> .
>
> "Objects of class "Class" are deserialized as the name of the class and fingerprint. A fingerprint is a hash of the interfaces, methods, and fields of the class. The "resolve-Class" method is called to find the class by name and return its "Class" object. The hash is computed for the returned class and compared with the hash of the class serialized. Deserialization proceeds only if the class matches. This ensures that the structure of the stream matches the structure of the class. All other normal Java classes are deserialized by reading the encoding of its Class followed by its fields."

Initially, Applicants respectfully submit that Murray does not teach or suggest providing an object with a hashtable which includes executable program code as a portion of the hashtable. In Murray, a fingerprint or hash of interfaces, methods and fields of the class is discussed, but there is no teaching or suggestion that executable program code per say becomes part of the hashtable included within the object.

Additionally, Applicants respectfully submit that a careful reading of Murray fails to uncover any teaching or suggestion that the logic of the executable program code employs as data input at least one other element of the hashtable. The above-cited lines from Murray discuss a fingerprint being a hash of the interfaces, methods and fields of the class, and the use of this fingerprint by comparing the hash for the returned class compared to the hash for the serialized class. Deserialization occurs only if the class fingerprints match. This teaching is not relavent to the language recited in Applicants' independent claims. In Applicants' invention, the executable program code included within the hashtable uses as data input at least one other element of the hashtable.

Based upon the above-noted defenciences of Murray when applied against Applicants' independent claims, it is respectfully submitted that the independent claims presented are not anticipated by Murray. Thus, reconsideration and withdrawal of the rejection based thereon are requested.

Jones describes a method and system for deterministic hashs to identify remote methods. When a client wishes to invoke a remote method located on a server, the client sends a hash value identifying the remote method to the server in the "remote method innovacation" (RMI) call. In one implementation, this hash value is created by applying a hash function to a method string name and the parameter type list and possibly return type. When the server receives the RMI call, the server identifies which method is being called using the received hash value. (See abstract of Jones.)

Cited against Applicants' independent claims are hash value 308, as well as column 9, lines 5-67 and column 10, lines 1-22 of Jones. As taught at column 9, lines 28-31 of Jones:

> "In one implementation, hash value 308 is a hash value resulting from applying a standard hash function to the combination of the method name and parameter type list 318 of the remote method 316, ..."

As clearly taught in the above cited lines of Jones, the Jones protocol applies a hash function to the *method name* and parameter type list. The method name is distinct from the executable program code itself, and is merely a name for code. In Applicants' invention, the executable program code itself is part of the hashtable being transferred from the sender computer process to the receiever computer process. Thus, the cited lines of Jones teach away from Applicants' recited invention.

For the above-noted reasons, Applicants respectfully submit that there is no anticipation of the independent claims presented based upon Jones. In Jones, the hash function operates on the method name, not executable program code.

Additionally, Applicants respectfully submit that the RMI protocol is a well established protocol. In RMI, it is assumed that the method to be invoked is already resident at the remote process, i.e., the server. In RMI, both the sender and receiver have a knowledge agreement of the methods to be invoked via a registering process. This is in contrast with Applicants' recited invention, wherein the executable program code is essentially "pushed" as part of the hashtable from the sender computer process to the receiver computer process. No similar teaching is provided by Jones.

Further, Applicants respectfully submit that the RMI protocol is well known in the art, and it would not have been obvious to one of ordinary skill in the art to modify RMI in some way in order to include executable program code within a hashtable in a matter such as recited by Applicants in the independent claims presented. To do so would render pointless the RMI protocol itself.

Wollrath also describes an RMI process wherein a remote method invocation is described using a generic proxy class. A client machine transmits a call for invocation of a method of a remote object including an identifier for the method object. A server machine receives the identifier and uses generic code to invoke the method object and return an indication of the invoked method along with any relevant parameters. (See abstract of Wollrath.)

Applicants respectfully submit that Wollrath does not anticipate the independent claims presented for the above-noted reasons with respect to the Jones patent. Steps 703-705 of FIG. 7 in Wollrath are cited in the Office Action as well as column 10, lines 41-50 & 61-67. However, Applicants respectfully submit that the *method hash* discussed at column 10 of Wollrath is described in the U.S. Patent Application entitled "Method and System for Deterministic Hashes to Identify Remote Methods" which is incorporated therein at column 10, lines 52-54. In this other patent (now U.S. Letters Patent 6,629,154) the method hash is taught as a technique of uniquely identifying a remote method to be invoked on a server which is *computed from the method signature* sent from the client to the server with the call request. When a client wishes to invoke a remote method located on a server, the client sends a hash value identifying the remote method to the server in the "remote method invocation" (RMI) call. Thus, again Wollrath is teaching the creation of a hash from the method *signature* which is sent from the client to the server. There is no teaching and suggestion in Wollrath of transferring within an object a hashtable which includes executable program code. Rather, in Wollrath, the RMI protocol is employed which again assumes the existence on the server (i.e., the remote process) of the program code (i.e., method) to be invoked.

For at least the above-noted reasons, Applicants respectfully submit that there is no anticipation of the independent claims presented based upon Wollrath. There is no teaching in Wollrath of executable program code per se being transferred as part of a hashtable from a sender computer process to a receiver computer process. As such, Applicants respectfully request reconsideration and withdrawal of the anticipation rejection based thereon.

For at least the above-noted reasons, Applicants respectfully submit that the independent claims presented patentably distinguish over Murray, Jones and Wollrath.

The dependent claims are believed allowable for the same reasons as the independent claims, as well as for their own additional characterizations.

For example, claims 4, 13 & 20 further characterize the respective independent claims by reciting protocol which includes adding data to the hashtable at the receiver computer process. The added data at the receiver computer process is indicated to be relevant to the executable program code and is *added prior to invoking of the executable program code using as data input only the hashtable.* Applicants respectfully submit that the careful reading of Wollrath fails to teach or suggest this protocol. Cited against these claims is step 706, as well as column 10, line 64 – column 11, line 7 of Wollrath. Step 706 in FIG. 7 of Wollrath states: "Server unmarshalls the parameters for the operation given the types of parameter specified in the method object." The cited lines of column 10 & 11 state:

> "...Server machine 606 unmarshals the parameters for the operation given the types of the parameters specified in the method object (step 706). Step 706 may involve building a method table, which compiles values for particular methods and is initialized when a remote object is created and exported. The generic code creates a correspondence between a method hash and a particular method object. Thus, by using the hashes in the method table, different skeletons typed to different types of objects is not required for method invocation correspond method hash to method object."

Applicants respectfully submit that the above-noted teachings from Wollrath are not relevant to the recited protocol. In Applicants dependent claims 4, 13 & 20 the recited invention includes *adding data to the hashtable at the receiver computer process.* There is no data taught in step 706 of FIG. 7 or the accompanying discussion in columns 10 & 11 being added to a hashtable received at a receiver computer process. As such, these claims are believed patentable over the applied art.

Still further, dependent claims 4, 13 & 20 recite that the executable program code *uses as data input only the hashtable.* Thus, the data input recited in claims 4, 13 & 20 is data provided by the sender computer process placed into the hashtable before sending of the object and data placed into the hashtable by the receiver computer process after receipt of object. Again, there is no similar teaching or suggestion in the applied art. For at least the above-noted reasons, Applicants respectfully request reconsideration and allowance of these claims.

Dependent claims 7 & 16 further indicate that the first hashtable received from the first sender computer process and the second hashtable received from the second sender computer process are merged at the receiver computer process into a common hashtable. In rejecting this subject matter, the Office Action relies upon Wollrath and Skinner. In particular, lines 34-51 of Skinner. These lines teach:

> "Serialization provides a useful mechanism for object persistence and transmission. With respect to persistence, a serialized object may be stored in memory any length of time and regenerated with the same state the object had at the time it was serialized. With respect to object transmission, a serialized object may be transmitted between remote clients and servers as a data streams or packets in accordance with known communication protocols, such as the protocol implemented by the Remote. Method Invocator (RMI) API. Also, serialized objects may be written to a shared portion of memory by one application and read out of the shared memory by another application. Client-side and server-side communication management components 305A and 305B implement methods to perform serialization and deserialization functions. Data transmission and reception of serialized objects, may be implemented using any known communication protocol as described above."

Applicants respectfully submit that the above-noted teachings from the Skinner patent are simply not relevant to the recited functionality. In Applicants recited invention, the hashtable includes executable program code to be invoked. In Skinner, objects and method calls are transmitted between a client and application server. Objects configured with metadata can be serialized, i.e. broken down into a set of data bytes and later reconstituted (i.e., deserialized by the same or different application) to generate a copy of the original object. It is respectfully submitted that this breaking down of objects configured with metadata refers to a datastream, not to a hashtable containing executable program code. There is no discussion in Skinner that the objects configured with metadata include executable program code within a hashtable. As such, Applicants respectfully submit that claims 7 & 16 patentably distinguish over Wollrath and Skinner taken together. Further, there is no teaching or suggestion in Skinner of iterating through the common hash table for executable program code to be invoked *using the common hashtable as the only data input thereto.* For at least these reasons, Applicants request reconsideration allowance of dependent claims 7 & 16.

Dependent claims 8 & 17 are believed allowable for the same reasons noted above respect to claims 7 & 16 as well as those reasons stated above respect to dependent claims 4, 13 & 20. These dependent claims further recite that the receiver computer process adds data to the common hashtable prior to invoking the executable program code using as data input only the common hashtable. In the Office Action, Skinner is cited against the subject matter of claims 8 & 17. In particular, column 16, lines 43-51 discuss serialized objects which may be written to a shared portion of memory in one application and read out of the shared memory by another application. Client side and server side communication management components implement to perform serialization and deserialization functions. These teachings are simply not relevant to the functionality recited in Applicants' dependent claims 8 & 17. There is no teaching or suggestion in Skinner (or in Wollrath) of a receiver computer process *adding data to a hashtable* which is relevant to the received executable program code within the hash table and which is added prior to invoking the executable program code *using as data input only the hashtable*. As such, Applicants respectfully submit that these claims patentably distinguish over the applied art.

All claims are believed to be in condition for allowance and such action is respectfully requested.

Should any issue remain unresolved, Applicants' undersigned representative requests a telephone interview with the Examiner to further discuss the matter in the hope of advancing prosecution of the subject application.

Kevin P. Radigan

Kevin P. Radigan, Esq.
Attorney for Applicants
Reg. No.: 31,789

Dated: July 27, 2007

HESLIN ROTHENBERG FARLEY & MESITI P.C.
5 Columbia Circle
Albany, New York 12203
Telephone: (518) 452-5600
Facsimile: (518) 452-5579